ProFlip: Targeted Trojan Attack with Progressive Bit Flips

Huili Chen Cheng Fu Jishen Zhao Farinaz Koushanfar
University of California, San Diego
Email: {huc044, cfu, jzhao, farinaz}@ucsd.edu

Abstract

*The security of Deep Neural Networks (DNNs) is of great importance due to their employment in various safety-critical applications. DNNs are shown to be vulnerable against the Trojan attack that manipulates model parameters via poisoned training and gets activated by the predefined trigger during inference. In this work, we present ProFlip, the first targeted Trojan attack framework that can divert the prediction of the DNN to the target class by progressively identifying and flipping a small set of bits in model parameters. At its core, ProFlip consists of three key phases: (i) Determining significant neurons in the last layer; (ii) Generating an effective trigger pattern for the target class; (iii) Identifying a sequence of susceptible bits of DNN parameters stored in the main memory (e.g., DRAM). After model deployment, the adversary can insert the Trojan by flipping the critical bits found by ProFlip using bit flip techniques such as Row Hammer or laser beams. As the result, the altered DNN predicts the target class when the trigger pattern is present in any inputs. We perform extensive evaluations of ProFlip on CIFAR10, SVHN, and ImageNet datasets with ResNet-18 and VGG-16 architectures. Empirical results show that, to reach an attack success rate (ASR) of over 94%, ProFlip requires only **12** bit flips out of 88 million parameter bits for ResNet-18 with CIFAR-10, and **15** bit flips for ResNet-18 with ImageNet. Compared to the SOTA, ProFlip reduces the number of required bits flips by **28**$\times \sim$ **34**$\times$ while reaching the same or higher ASR.*

1. Introduction

Deep Neural Networks (DNNs) have empowered a paradigm shift in various real-world applications due to their unprecedented performance on complex tasks. The deployment of DNNs in safety-critical fields such as biomedical diagnosis, autonomous vehicles, and intelligent transportation [23, 26, 38] renders model security crucial. Prior works have demonstrated the vulnerability of DNNs against a diverse set of attacks. For instance, adversarial samples are strategically crafted inputs that look normal to human beings while they can mislead the model to produce wrong outputs during inference [13, 20, 43]. Data poisoning is a training-time attack that tampers with model weights by injecting incorrectly labeled data into the training set [7, 28]. Neural Trojan [25, 15, 24] is a targeted attack that manipulates both the model parameters and the inputs (i.e., adding the trigger). In this work, we focus on Trojan attacks and aim to design an efficient approach for Trojan insertion without poisoned training.

A typical neural Trojan attack has two essential subroutines: *trigger generation* and *Trojan insertion* [15, 24]. The trigger is a specific pattern in the input space that controls Trojan activation (e.g., a white square at the image corner). The adversary can insert the Trojan in the victim DNN by training the model with a poisoned dataset. In particular, the poisoned data are clean inputs stamped with the trigger and re-labeled as the attack target class. Trojan attacks have two goals: *effectiveness* and *stealthiness*. Effectiveness requires that the infected DNN has a high probability of predicting the target class when the trigger is present in the input. Stealthiness requires the Trojaned model to produce correct outputs on clean data.

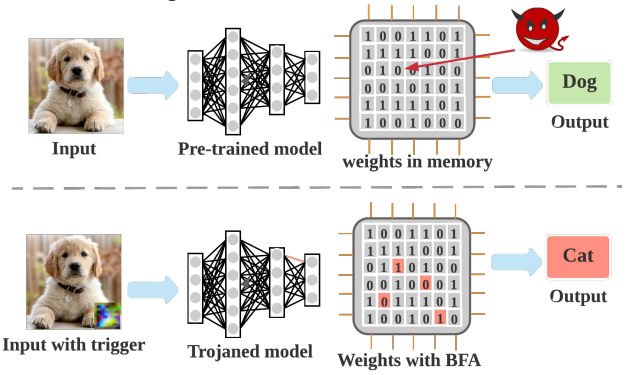

Figure 1: Demonstration of the proposed ProFlip attack. The top part shows normal inference of a clean model whose weights are subject to bit flip attacks. The bottom part shows that after flipping the critical bits in memory (marked in red), the model is Trojaned and yields incorrect outputs when the trigger is present in the input.

Existing Trojan attacks assume that the adversary is the model developer (e.g., cloud server) who has sufficient computing power for DNN training. The victims are end-users that obtain the pre-trained models from the third-party

providers. Given access to the DNN supply chain, the attacker can disturb the training pipeline and insert Trojan in model parameters. Recently, a line of research has demonstrated parameter manipulation attacks against DNNs using *bit flip techniques* such as Row Hammer [18, 37] and laser beams [6, 10]. *Bit Flip Attacks (BFA)* [17, 32, 31] eliminate the requirement of training access in previous Trojan attacks [15, 24], thus posing a strong runtime threat to DNNs after model deployment.

ProFlip Overview. In this paper, we present ProFlip, an innovative bit flip-based Trojan attack that inserts the Trojan into a *quantized* DNN by altering only a few bits of model parameters stored in memory (e.g., DRAM). Figure 1 illustrates the working mechanism of ProFlip attack against the DNN after its deployment.

Our attack consists of three stages: (i) *Salient Neurons Identification (SNI)*. We use forward derivative-based *saliency map* to identify neurons important for the target class in the last layer. (ii) Trigger generation. ProFlip generates the trigger pattern that can fool the DNN to the target class and stimulates salient neurons to large values simultaneously. (iii) *Critical Bits Search (CBS)*. ProFlip gradually/sequentially pinpoints the most vulnerable parameter bits of the victim DNN in a greedy manner. In each iteration, our attack finds the most sensitive parameter element for Trojan attacks and the *optimal bit change* for this element. ProFlip determines the sequence of bit flips to ensure that the Trojaned DNN has a comparable accuracy as the benign model on clean data, while predicts the target class when the trigger is present in inputs. Our evaluation results show that ProFlip only requires **12** bit flips out of 88 millions to achieve an ASR of 94% for ResNet-18 with CIFAR10, and **15** bit flips for ResNet-18 with ImageNet.

2. Preliminaries and Related Works

Inducing Bit Flips in Memory Storage. Memory storage components such as DRAM chips are indispensable for computing systems [12, 21]. The susceptibility of commercial DRAMs to *disturbance errors* has been demonstrated by Kim *et.al* in [18]. The paper finds out that repeatedly accessing a DRAM row can corrupt data stored in neighboring rows, i.e., causing bit flips '0' $\rightarrow$ '1' or '1' $\rightarrow$ '0'. This disturbance error in DRAMs is called *Row Hammer Attack (RHA)* [18, 37]. The root cause of RHA is that frequent row activation results in voltage fluctuations, which leads to the charge loss of adjacent rows. Furthermore, the adversary can perform precise bit flip at any desired location by profiling the DRAM memory layout [42]. RHAs pose severe security threats to the computing platforms since they can evade common data integrity checks and error correction techniques [29, 14]. Besides RHA, laser fault injection can also induce single bit flip in hardware [6, 10].

Background on Quantized DNN (QNN). Model quantization is a widely-deployed technique that uses fixed-point representation to improve the efficiency of DNN inference [34, 22, 41]. The weight parameter of a layer in a N-bit quantized DNN is represented and stored as a signed integer in two's complement format, i.e., $\mathbf{b} = [b_{N-1}, ..., b_0] \in \{0, 1\}^N$. In this work, we adopt uniform weight quantization scheme that is identical to the TensorRT technique [27]. To train QNNs that contain non-differential stair-case functions, we apply straight-through estimator as suggested in prior works [46, 32]. For l^{th} layer of the QNN, the binary vector $\mathbf{b}$ can be converted into a fixed-point real number:

$$W_l = (-2^{N-1} \cdot b_{N-1} + \sum_{i=0}^{N-2} 2^i \cdot b_i) \cdot \Delta_l, \qquad (1)$$

where Δ_l is the step size of the weight quantizer for layer l. Note that for a pre-trained QNN, the step size of each layer is a known constant and can be computed based on the maximum absolute parameter value and quantization bitwidth:

$$\Delta_l = \frac{max(abs(W_l))}{2^{N-1} - 1} \qquad (2)$$

Existing Bit Flip Attacks on DNNs. Recently, bit flip attacks have been demonstrated to divert the DNN by manipulating the bit representation of model parameters [17, 32]. We categorize BFAs into two types based on the attack model: Adversarial Weight Attack (AWA) and Trojan attack. AWA only modifies specific weight bits and keeps the input sample unchanged, while Trojan attacks require modification of both DNN parameters and input data (i.e., adding the trigger). AWAs can be untargeted [17, 31] or targeted [33, 8]. Terminal brain damage [17] demonstrates the first untargeted BFA on floating-point DNNs where the vulnerable bits are found by simple heuristics. The paper [31] proposes an untargeted BFA on fixed-pointed DNNs by searching weight bits with large gradient magnitudes in an iterative in-layer and cross-layers way. The authors extend this idea and present a targeted attack variant in [33]. The paper [8] formulates the targeted adversarial weight attack as a binary integer programming problem and solves it with Alternating Direction Method of Multipliers.

To the best of our knowledge, TBT [32] is the only BFA that performs bit flip-based Trojan attacks on quantized DNNs. TBT deploys Neural Gradient Ranking (NGR) to find susceptible neurons and generates the Trojan trigger using Fast Gradient Sign Method (FGSM). For Trojan insertion, TBT uses gradient descent to finetune the weight bits associated with the neurons found by NGR.

Limitation of Prior Works. Our work falls into the same category as the TBT attack [32]. However, TBT is *impractical* in the real-world setting since it requires a large number of bit flips. For instance, to achieve an ASR of 93.2%, TBT requires to flip 413 bits of ResNet-18 with CIFAR-10 dataset. This is because TBT updates a pre-defined number of weight elements ($w_b = 150$ in [32]) in the last layer

to minimize the Trojan insertion loss. Such a formulation is *oblivious* of the BFA overhead in terms of the required number of bit flips. Furthermore, TBT does not provide insights on attack parameter selection. Only the parameter of the last layer is considered for Trojan bits search. Our empirical results in Section 4 show that the last layer of the DNN is not necessarily the optimal target for BFA.

Threat Model. Consistent with prior works [32], we assume that the adversary knows the architecture and weights of the victim DNN. Besides this, the attacker also knows the memory allocation of model parameters. This is essential to perform precise bit flips at the desired locations. Furthermore, we assume the attacker has a small set of clean data samples. To activate the inserted Trojan, the attacker shall add the pre-defined trigger in the input during inference. Note that our attack does not require information of the training data or access to the training pipeline.

3. ProFlip Methodology

ProFlip is motivated to address the efficiency and effectiveness limitations of TBT [32] for critical bits search. We propose a systematic attack framework that *progressively* identifies a sequence of vulnerable parameter bits for Trojan attacks. ProFlip consists of three key stages as illustrated in Figure 2. We introduce each stage in the sections below.

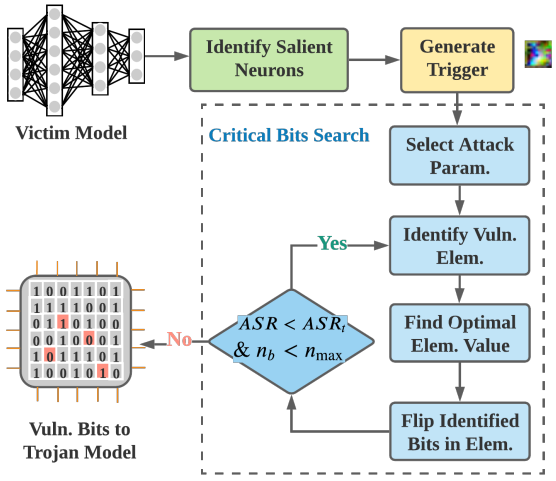

Figure 2: Global flow of ProFlip. Given a victim model, we first identify salient neurons associated with the target class. Trigger is then generated to control Trojan activation. Finally, ProFlip performs iterative critical bits search to identify vulnerable bits in the model parameters.

3.1. Salient Neurons Identification (SNI)

In the first stage, ProFlip identifies neurons important for the targeted Trojan attack using the idea of *adversarial saliency map* [30]. Particularly, our attack leverages the *forward derivative*-based saliency map construction, which is also known as Jacobian Saliency Map Attack (JSMA) [30, 40]. Algorithm 1 outlines the procedures of ProFlip's SNI method. M_L and $M_{1:L-1}$ denote the last layer of M and the model without the last layer, respectively. The key step of SNI is computing the saliency map (line 9), which returns the top-2 coefficients in the search space (Γ) that maximize the saliency map. Without the loss of generality, ProFlip considers increasing the values of the searched features (i.e., $\theta > 0$) for targeted attack.

Algorithm 1 Salient neurons identification using adversarial saliency map

INPUT: Victim DNN (M) of L layers, target class t, a small set of clean data of size S ($D = \{X, Y\}$), perturbation added to each feature per step (θ), maximum fraction of perturbed features (γ).

OUTPUT: Indices of significant neurons in the last layer of the DNN (I_t).

1: **for** $0 < i < S$ **do**
2: Obtain activation map: $\mathbf{a}^0_{L-1} \leftarrow M_{1:L-1}(X_i)$
3: Initialize: $\mathbf{a}^* \leftarrow \mathbf{a}^0_{L-1}$, $\Gamma = \{1, ..., |\mathbf{a}^*|\}$, $I_i = [\,]$
4: Value range: $a_{max},\ a_{min} = max(\mathbf{a}^*),\ min(\mathbf{a}^*)$
5: **while** $M_L(\mathbf{a}^*) \neq t\ \ \&\ \| \delta_{\mathbf{a}} \| < \gamma$ **do**
6: $p_1,\ p_2 = saliency\_map(\nabla M_L(\mathbf{a}^*),\ \Gamma,\ t)$
7: Modify p_1 and p_2 in $\mathbf{a}^*$ by θ
8: Remove p_1 from Γ if $\mathbf{a}^*(p_1) \notin [a_{min}, a_{max}]$
9: Remove p_2 from Γ if $\mathbf{a}^*(p_2) \notin [a_{min}, a_{max}]$
10: Update: $I_i.add(p_1, p_2)$, $\delta_{\mathbf{a}} = \mathbf{a}^* - \mathbf{a}^0_{L-1}$
11: $I_t = find\_intersection(I_0,\ ...,\ I_{S-1})$
12: **return** I_t

3.2. Trojan Trigger Generation

We consider *physically realizable* trigger patterns in this work. Particularly, we consider an attack scenario that the adversary can 'stamp' the input with a pre-defined trigger pattern (i.e., pixel values are replaced by the trigger in a constrained region). This type of trigger is effective in practice and has been widely used in previous Trojan attacks [4, 15, 9, 39]. Trigger injection can be characterized by a generic function $A(\cdot)$ with three variables: clean input $\mathbf{x}$, trigger mask $\mathbf{m}$, and trigger values $\mathbf{\Delta}$:

$$\mathbf{x}^* = A(\mathbf{x},\ \mathbf{m},\ \mathbf{\Delta}),$$
$$\mathbf{x}^*_{w,h,c} = (1 - \mathbf{m}_{w,h}) \cdot \mathbf{x}_{w,h,c} + \mathbf{m}_{w,h} \cdot \mathbf{\Delta}_{w,h,c}, \quad (3)$$

where w, h, and c denote the width, height, and color channel dimension, respectively. The mask of a physical trigger ($\mathbf{m}$) is a 2D binary matrix shared across color channels.

The objective of ProFlip's trigger generation is two-fold: (i) With the salient neurons I_t identified in the SNI stage, the trigger is expected to stimulate these neurons to large values; (ii) When the physical trigger is applied on clean inputs, the DNN shall predict the target class t. These two goals are formulated as two adversarial loss terms below:

$$\mathcal{L}_{mse}(M_{1:L-1}(A(\mathbf{x},\ \mathbf{m},\ \mathbf{\Delta}));\ c), \quad (4)$$
$$\mathcal{L}_{ce}(M(A(\mathbf{x},\ \mathbf{m},\ \mathbf{\Delta}));\ t). \quad (5)$$

Here, the target value for salient neurons c is a large constant selected by the adversary. Note that c is positive since ProFlip's SNI stage employs a positive step size. We use Mean Square Error (MSE) and Cross-Entropy (CE) loss functions to compute these two loss terms, respectively.

ProFlip formulates trigger generation as an optimization problem and solves it using gradient descent:

$$\mathcal{L}_{trig} = \lambda_1 \cdot \mathcal{L}_{mse}(\mathbf{x}, \mathbf{m}, \mathbf{\Delta};\ c) + \lambda_2 \cdot \mathcal{L}_{ce}(\mathbf{x}, \mathbf{m}, \mathbf{\Delta};\ t) \quad (6)$$

$$\min_{\mathbf{m}, \mathbf{\Delta}} \mathcal{L}_{trig}(\mathbf{x}, \mathbf{m}, \mathbf{\Delta}) \text{ for } \mathbf{x} \in \mathbf{X}, \quad (7)$$

where λ_1 and λ_2 are two hyper-parameters that control the weights of two loss terms in $\mathcal{L}_{trig}$.

3.3. Critical Bit Search (CBS)

Bit Search Formulation. Given a victim model M, salient neurons I_t and the trigger $\mathbf{\Delta}$, we aim to find a few bits of model parameters such that when these bits are flipped, the infected model M^* has a high Trojan ASR on poisoned inputs $\mathbf{x}^*$. Mathematically, $Prob[M^*(\mathbf{x}^*) = t]$ shall be large. We define the loss of critical bits search as follows:

$$\mathcal{L}_{CBS} = \sum_{\mathbf{x}} \mathcal{L}_{mse}(M^*_{1:L-1}(\mathbf{x}^*); c) + \mathcal{L}_{ce}(M^*(\mathbf{x}^*); t). \quad (8)$$

Challenges. A high-performance DNN has a tremendous amount of parameters [16, 36, 11]. For instance, VGG-16 has 138 million parameters and the 8-bit quantized variant needs $1,104$ million bits for storage [35]. It has also been shown that randomly flipping a limited number of bits in the quantized DNN yields a very low attack success rate [31]. The large search space makes the exhaustive search of critical bits infeasible. As such, developing an efficient and effective bit search algorithm is difficult.

Our Intuition. ProFlip addresses the challenges of critical bits search by *shrinking* the search space progressively. In particular, our attack starts with the highest abstraction level (*which parameter* in which layer to attack), then proceeds to a more fine-grained level (*which element* in this parameter to attack), and finally determines the lowest bit level (what is the *optimal value* of this element). Such a progressive approach allows us to constrain the number of bit flips (n_b) to a very small value while ensuring a high ASR.

CBS Workflow. ProFlip's CBS starts with attack parameter selection (S1), which is a *one-time*, offline process. Then, a sequence of vulnerable bits is identified in an *iterative* way. In each iteration, the current most vulnerable element is identified (S2) and its optimal value is determined (S3). The corresponding bits in this element are then flipped to reach the optimal value and CBS proceeds to the next iteration. Our CBS pipeline terminates when the desired ASR or the maximal number of allowed bit flips is reached. Algorithm 2 shows the procedure of ProFlip's critical bits search. We detail three key steps (S1 $\sim$ S3) of CBS below.

(S1) Attack Parameter Selection. ProFlip's CBS first performs *parameter-level sensitivity analysis* to determine the most vulnerable parameter. To this end, we introduce a new metric to characterize the influence of a parameter on Trojan attacks. For a parameter in a QNN, we define its *fitness score* F as the product of the gradient magnitude and the maximal allowed value change. The rationale behind this definition is that: (i) Gradient magnitude of a parameter regarding $\mathcal{L}_{CBS}$ is a direct measurement of its sensitivity; (ii) BFAs intend to modify vulnerable parameters to large values [32, 31, 17]. To maintain the quantization step size Δ_l after bit flips, the perturbation allowed on the original parameter (W_l) is *bounded*. Mathematically, we need to ensure $max(abs(W_l)) = max(abs(W_l^*))$ where W_l^* is the perturbed parameter. As such, our fitness definition incorporates this maximal value change.

Algorithm 2 ProFlip's workflow of critical bits search.

INPUT: Victim DNN (M), target class t, trigger pattern $\{\mathbf{m},\ \Delta\}$, a small set of clean data ($D = \{X, Y\}$), target ASR (ASR_t), maximal allowed bits flips n_{max}).
OUTPUT: A sequence of bit flips for Trojan attack.

Initialize: $n_b = 0$, $n_e = 0$, $\mathbf{s_b} = [\,]$, ASR=0
$p_{sens} \leftarrow select\_attack\_param(M,\ D)$
while $ASR < ASR_t$ and $n_b < n_{max}$ **do**
  $elem \leftarrow identify\_vuln\_elem(M, p_{sens}, D)$
  $elem^* \leftarrow find\_optim\_value(M, p_{sens}, elem, D)$
  $\mathbf{f} \leftarrow compute\_bit\_flips(elem, elem^*)$
  $\mathbf{s_b}.add(\mathbf{f}), n_b += |\mathbf{f}|, n_e += 1$
  $ASR \leftarrow eval\_Trojan\_attack(M,\ \mathbf{s_b},\ D)$
return $\mathbf{s_b}$

Algorithm 3 Parameter-level sensitivity analysis.

INPUT: Victim DNN (M) with parameters P, target class t, trigger pattern $\{\mathbf{m},\ \Delta\}$, a small set of clean data ($D = \{X, Y\}$), maximal magnitude of parameters in quantization Q.
OUTPUT: Index of the most vulnerable parameter.

1: Compute CBS loss $\mathcal{L}_{CBS}$
2: **for** $\mathbf{p} \in P$ **do**
3: Compute partial derivative $\frac{\partial \mathcal{L}_{CBS}}{\partial \mathbf{p}}$
4: **for** $elem \in \mathbf{p}$ **do**
5: **if** $\frac{\partial \mathcal{L}_{CBS}}{\partial \mathbf{p}}|_{elem} < 0$ **then**
6: $step \leftarrow Q_{\mathbf{p}} - elem$
7: **else**
8: $step \leftarrow 0$
9: Fitness $F(p, elem) = abs(\frac{\partial \mathcal{L}_{CBS}}{\partial \mathbf{p}}|_{elem}) \cdot step$
10: Optim. attack parameter: $p_{sens} = \underset{p}{\text{argmax}}\, F(p,\ elem)$
11: **return** p_{sens}

Algorithm 3 outlines the detailed procedures of ProFlip's parameter sensitivity analysis. The key step is computing the fitness score of model parameters in line 9. In this work,

we only consider parameters with *negative gradients* (line 5) based on the empirical observations that bit flips that change parameters to extremely large numbers are more effective than decreasing them [17, 32, 31].

(S2) Vulnerable Element Identification. After ProFlip performs parameter-wise attack sensitivity analysis in (S1), we tackle a more fine-grained problem, i.e., which scalar element in the parameter is most favorable for the Trojan attack. This *progressive vulnerability locating* paradigm is beneficial for minimizing the final number of bit flips. ProFlip leverages the fitness score computed in (S1) to characterize each element in the identified sensitive parameter (p_{sens}). The adversary can determine the most vulnerable element in the parameter p_{sens} as follows:

$$elem\_loc = \underset{elem}{\operatorname{argmax}} F(p_{sens},\ elem). \tag{9}$$

(S3) Optimal Value of Element. Recall that the Trojan attack needs to be both stealthy and effective, we define the Trojan injection loss as follows:

$$\mathcal{L}_{troj} = \gamma_1 \cdot \mathcal{L}_{ce}(M^*,\ D) + \gamma_2 \cdot \mathcal{L}_{CBS}, \tag{10}$$

where $\mathcal{L}_{CBS}$ is defined in Eq. (8). γ_1 and γ_2 are two hyper-parameters that control the trade-off between Trojan stealthiness and efficacy.

With the attack location information ($p_{sens},\ elem\_loc$) found in (S1) and (S2), the remaining question to launch BFA is to find which bits in the binary representation of this scalar element to flip. Equivalently, we can find the optimal value for this particular element. This question can be mathematically formulated as follows:

$$\mathbf{b} = quantize(M(p_{sens},\ elem\_loc)),$$
$$\mathbf{b}^* = bits\_flips([b_{N-1}, ..., b_0],\ \mathbf{m_b}), \tag{11}$$
$$\mathbf{m_b}^* = \underset{m_b}{\operatorname{argmin}}\, \mathcal{L}_{troj}(M^*,\ \mathbf{\Delta};\ D), \tag{12}$$

where $\mathbf{m_b}$ is the bit mask vector that determines which bits in $\mathbf{b}$ shall be flipped to minimize $\mathcal{L}_{troj}$. Since $\mathbf{m_b}$ is a discrete variable, using gradient descent to solve the optimization problem in Eq. (12) is infeasible. One can enumerate all possible bit masks and select the one that results in the lowest $\mathcal{L}_{troj}$ with a computation complexity of $\mathcal{O}(2^N)$.

Alternatively, ProFlip provides a *complexity controllable* solution using grid search. More specifically, our attack divides the feasible value range of parameter p_{sens} ($[-R, R]$ where $R = max(abs(p_{sens}))$) into K parts and evaluates $\mathcal{L}_{troj}$ on these K partitioning points. The cut point with the smallest loss is used as the approximate optimal value $elem^*$ for the identified element. Once $elem^*$ is determined, its binary representation $\mathbf{b}^*$ and the corresponding bit mask ($\mathbf{m_b}^*$) can be computed. Finally, the required number of bit flips (n_b) is calculated as the Hamming Distance (HD) between the two binary strings:

$$n_b = Hamming\_Distance(\mathbf{b},\ \mathbf{b}^*). \tag{13}$$

Note that ProFlip's critical bits search is an iterative process as shown in Algorithm 2. The final bit flip sequence is the sequential aggregation of results in all iterations.

Bit Trojan Activation. As shown in Figure 2, the vulnerable bits are identified when CBS terminates. The attacker then deploys bit flip techniques such as Row Hammer [18, 29, 37] to modify these critical bits in memory. Meanwhile, he shall apply the trigger designed in Section 3.2 on the input of his interests to activate the Trojan.

4. Evaluation Results

4.1. Experimental Setup

Datasets and Architectures. We investigate the attack performance of ProFlip on three datasets used in TBT [32]: CIFAR-10 [1], SVHN [3], and ImageNet [2]. The first two datasets have 10 classes and image dimension $32 \times 32 \times 3$, while ImageNet has 1000 classes and input dimension $224 \times 224 \times 3$. We assume the adversary has a clean data batch (taken from the training set) of size 256 in all experiments. Consistent with the prior work [32], we evaluate our attack on two types of model architectures: ResNet-18 and VGG-16, with a quantization level of 8-bit. We investigate ProFlip's performance across all benchmarks in the majority of the experiments and select ResNet-18 with CIFAR-10 as an exemplar in our ablation study (Section 4.4).

Evaluation Metrics. We use Test Accuracy (TA) after Trojan insertion (i.e., critical bits flipping) to measure attack stealthiness. To assess attack efficacy, we use the attack success ratio (the percentage of inputs that are mispredicted by the Trojaned model as the target class when the trigger is applied) as the metric. For trigger generation, we use Trigger Area Percentage (TAP) to quantify the proportion of input replaced by the trigger [32]. To characterize the *efficiency* of our bit flip attack, we measure the total number of bit flips (n_b) to reach a particular ASR. The total number of elements changed (n_e) is also measured. We emphasize that we assess ProFlip's ASR on *unseen* inputs from the test set, thus to corroborating its *generalized effectiveness*.

ProFlip Configuration. For salient neurons identification, we set default parameters as $\theta = 0.1$, $\gamma = 0.5$ and target class $t = 2$ in Algorithm 1 for all benchmarks. For trigger generation, we use the same configuration as TBT [32] where the trigger is a *square* pattern with a pre-defined size locating at the bottom right of the image (i.e., trigger mask $\mathbf{m}$ is known). Therefore, the optimization problem in Eq. (7) only solves for the trigger value $\mathbf{\Delta}$. The hyper-parameters are set to $\lambda_1 = \lambda_2 = 1$ and $c = 10$ in $\mathcal{L}_{trig}$. We use a default trigger area $TAP = 9.76\%$ for experiments on CIFAR-10 and SVHN, and $TAP = 10.62\%$ on ImageNet. For critical bits search, the thresholds are set to $ASR_t = 94\%$ and $n_{max} = 100$ by default. The partitioning number for grid search is set to $K = 20$ in all experiments. When computing $\mathcal{L}_{troj}$ in Eq. (10), we use a fixed value of

γ_2=1 and set γ_1 such that $\gamma_1 \mathcal{L}_{ce} \sim 0.1 \cdot \gamma_2 \mathcal{L}_{CBS}$.

Baseline Attack. We use TBT [32] as our baseline attack since this is the only work that has the same attack objective and scenario as ProFlip. For quantitative comparison, we use the open-sourced implementation of TBT [5] and the configuration suggested in the paper [32].

4.2. Attack Effectiveness

End-to-end attack results. Table 1 summarizes ProFlip's performance on all benchmarks. The third and fourth columns show the test accuracy of the victim DNN before and after our attack. For ImageNet, TA is the top-1 test accuracy. The vulnerable parameter identified by ProFlip (p_{sens}) is shown in the sixth column. The total number of elements changed (n_e) and the total number of bit flips are given in the last two columns of Table 1. It can be seen that ProFlip achieves a high ASR (over 94%) while preserving the accuracy on clean data (test accuracy drop within $\sim 3\%$) across all benchmarks, thus satisfying the *stealthiness* and *effectiveness* criteria of Trojan attacks.

Table 1: Summary of ProFlip's performance. The target class is set as $t = 2$ in all cases. Trigger area $TAP = 9.76\%$ on CIFAR-10 and SVHN, and 10.62% on ImageNet.

Dataset	Model	Test Acc.(%) Before	Test Acc.(%) After	ASR	p_{sens}	n_e	n_b
CIFAR-10	ResNet-18	93.1	90.3	97.9	62	2	12
	VGG-16	89.7	88.1	94.8	45	3	16
SVHN	VGG-16	98.6	95.3	94.5	45	5	20
ImageNet	ResNet-18	69	67.6	94.3	60	3	15

Results of SNI. Recall that ProFlip starts with salient neurons identification. For ResNet-18 model where the second to the last layer (M_{L-1}) has 512 neurons, ProFlip saliency map-based method identifies 30 and 36 significant neurons on CIFAR-10 and ImageNet, respectively. For VGG-16 model where layer M_{L-1} has 4096 neurons, ProFlip's SNI finds 35 and 154 salient neurons on CIFAR-10 and SVHN, respectively. To validate the effectiveness of our SNI method, we measure the ASR of the model when the salient neurons (I_t) are set to the pre-specified large value $c = 10$ while other neurons are set to random values within the range of $[a_{min}, a_{max}]$. Empirical results show that ProFlip achieves ASR of 100% on all benchmarks.

Results of trigger generation. We employ SGD with a learning rate of 0.1 and train the trigger $\mathbf{\Delta}$ in Eq. (7) for 100 epochs. The batch size is 128 for ResNet-18 with CIFAR-10, and 64 for the other benchmarks. Table 2 shows the results of our trigger generation method where the ASR quantifies the efficacy of the trigger. Note that the trigger's ASR is the initial ASR for ProFlip's critical bits search, thus an effective trigger helps to reduce n_b for the desired ASR.

Results of parameter-level sensitivity analysis. We implement Algorithm 3 and show the results (p_{sens}) in Table 1. For ResNet-18 with CIFAR-10 and ImageNet, both $p_{sens} = 62$ and $p_{sens} = 60$ correspond to the weight parameter of the model's last dense layer. For VGG-16 with CIFAR-10 and SVHN, $p_{sens} = 45$ corresponds to the bias vector of the second to the last convolution layer in the model. Figure 3 shows the comparison results where two different parameters, $p_{sens} = 45$ (found by ProFlip) and $p_{sens} = 60$ (found by TBT [32]) are used by our BFA. One can see that our parameter-level sensitivity analysis successfully identifies the vulnerable parameter that allows the BFA to reach a high ASR in a few iterations (marked by the curve with stars), thus helps to reduce the number of bit flips n_b.

Table 2: Effectiveness of ProFlip's trigger generation. The target class is set to $t = 2$ for all benchmarks.

Dataset	Model	TAP (%)	ASR (%)
CIFAR-10	ResNet-18	9.76	50.96
	VGG-16	9.76	84.63
SVHN	VGG-16	9.76	83.46
ImageNet	ResNet-18	10.62	44.22

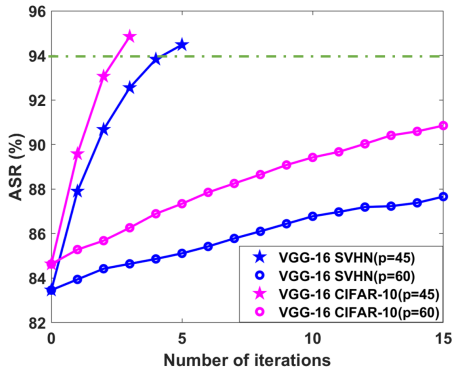

Figure 3: ProFlip's performance when different parameters are selected for attack. The curve color and the marker denote the benchmark and the selected parameter, respectively. The dashed line denotes the ASR threshold.

Results of critical bits search. To illustrate the *progressive* nature of ProFlip's CBS, we measure the ASR of our attack as the iteration proceeds. Figure 4 illustrates the *evolving* attack effectiveness of ProFlip on the benchmarks in Table 1. Note that ProFlip modifies a single element in each iteration (see Algorithm 2), thus the total number of elements changed n_e in Table 1 is the same as the final number of iterations shown in Figure 4. It can be observed that ProFlip's critical bits search effectively improves the ASR in the iterative process and converges in a few iterations.

4.3. Comparison with Prior Works

In this section, we compare the performance of ProFlip with the only existing counterpart: TBT [32]. For both attacks, we set the Trojan target class as $t = 2$. The trigger area percentage is set to $TAP = 10.62\%$ for the ImageNet benchmark and $TAP = 9.76\%$ in other cases. Besides

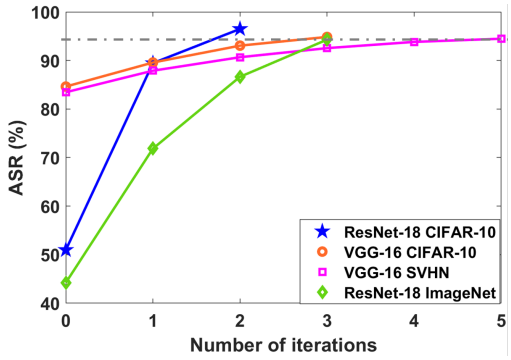

Figure 4: Performance of ProFlip's progressive critical bits search. The most vulnerable parameter (p_{sens}) and TAP for each benchmark are shown in Table 1. The dashed line denotes the termination condition $ASR_t = 94\%$.

using $ASR_t = 97\%$ for ImageNet experiment to match TBT, other parameters of ProFlip are the same as the ones used for Table 1. TBT reports multiple attack outcomes on ResNet-18 model with CIFAR-10 since different hyper-parameters w_b are used [32]. For a fair comparison, we report the result of TBT when it achieves the same level of test accuracy and ASR as ProFlip on the Trojaned model. Table 3 summarizes the performance comparison results across all benchmarks.

Table 3: Performance comparison between ProFlip and TBT [32]. For both attacks, the target class is set to $t = 2$.

Dataset	Model	TA (%)		ASR (%)		n_b	
		Ours	TBT	Ours	TBT	Ours	TBT
CIFAR-10	ResNet-18	90.3	89.1	97.9	93.2	12	413
	VGG-16	88.1	86.1	94.8	93.5	16	557
SVHN	VGG-16	95.3	73.9	94.5	73.8	20	565
ImageNet	ResNet-18	68.3	69.1	97.4	99.9	19	568

There are two observations from Table 3: (i) ProFlip is effective and *efficient*. Our attack reduces the number of bit flips n_b by an average of $\mathbf{31.8}\times$ compared to TBT, thus is more practical and threatening. (ii) ProFlip is more *generally effective and stealthy* across different datasets and model architectures compared to TBT. For VGG-16 model with SVHN dataset, TBT [32] can only achieve an ASR of 73.8% while flipping more than 500 bits. This large parameter change also leads to a test accuracy drop of 25.7%, which may reveal the Trojan attack. ProFlip achieves an ASR of 94.5% with 3.3% test accuracy drop by flipping only 20 bits. The root cause of TBT's deficiency on the SVHN benchmark is its incorrect selection of the vulnerable parameter. We show in Figure 3 that the BFA achieves a higher ASR with $p_{sens} = 45$ (found by ProFlip) compared to $p_{sens} = 60$ (found by TBT), suggesting the importance of attack parameter selection.

4.4. Ablation Study

Sensitivity to Target Class. We investigate the vulnerability of different target classes (TC) against ProFlip attack. Table 4 shows the evaluation results on ResNet-18 model with CIFAR-10 dataset. We use the same attack parameters ($\lambda_1 = \lambda_2 = 1$, $\gamma_1 = 2$, $\gamma_2 = 1$, $TAP = 9.76\%$) besides varying the target class t in this set of experiments. One can see that the most susceptible class of ResNet-18 is $t = 6$ where we only need to modify a single parameter ($n_e = 1$) by flipping 4 bits ($n_b = 4$). While the susceptibility varies with different target classes, ProFlip is generally effective (high ASR) and efficient (low n_b) in all attack scenarios.

Table 4: Vulnerability analysis of different target classes on ResNet-18 with CIFAR-10. The trigger area is 9.76% in all cases. Both TA and ASR are measured in percentage (%).

TC	TA	ASR	n_e	n_b	TC	TA	ASR	n_e	n_b
0	90.1	96.3	3	10	5	86.7	94.3	3	9
1	91.1	94.8	3	7	6	92.21	96.9	1	4
2	90.9	94	2	12	7	89	94.7	2	6
3	89	96.2	3	12	8	91.4	95.2	2	5
4	87.8	95.2	3	11	9	90.5	97.3	2	5

Sensitivity to Trigger Area. The trigger area has a direct impact on the ASR of ProFlip's trigger generation, thus also influences the critical bits search in the next stage. We vary the size of the square trigger while keeping the other hyper-parameters unchanged. Table 5 illustrates how ProFlip performance changes with the trigger area. We measure the ASR after trigger generation and critical bits search to show the impact of TAP on each stage. It can be seen that a larger TAP results in a higher ASR of trigger generation. This is because the trigger dimension increase, thus providing a larger optimization space when solving $\mathbf{\Delta}$ in Eq. (7).

Table 5: Effect of trigger area on ProFlip when attacking ResNet-18 model with CIFAR-10 dataset (target class $t = 2$). The column 'TrigGen' and 'CBS' denote trigger generation and critical bits search, respectively.

TAP (%)	TA (%)	ASR (%)		n_e	n_b
		TrigGen	CBS		
6.25	91.40	25.65	91.13	9	42
7.91	89.61	34.54	94.10	2	8
9.76	89.80	50.96	96.5	2	12
11.82	92.32	66.3	96.6	1	3

We can also observe that the total number of modified elements n_e (which is also the number of attack iteration) varies with the trigger area. This is due to the fact that a higher ASR inherited from the trigger generation stage provisions a better initialization for ProFlip's critical bits search, thus helps to reduce n_e. Note that a smaller value of n_e does not guarantee a smaller n_b, since the required number of bit flips in each element ($|\mathbf{f}|$ in Algorithm 2) is different. This fact is validated in the second and the third rows of Table 5 where $n_e = 2$ in both cases. When $TAP = 7.91\%$, ProFlip sequentially modifies two elements by flipping 3 and 5 bits in each element ($n_b = 8$), respectively. In the case where $TAP = 9.76\%$, ProFlip changes two elements

by flipping 5 and 7 bits in each element ($n_b = 12$).

Sensitivity to Attack Sample Size. Our threat model assumes the adversary has a small set of clean data sample D to assist the attack design. We investigate how ProFlip's performance changes when the size of the available data varies. Table 6 shows the experimental results on ResNet-18 model with CIFAR-10 dataset. We use the default configurations of ProFlip in this experiment. One can see that our attack is effective even when as few as 64 images are available. In general, a larger sample size is beneficial to ensure higher test accuracy and higher ASR for the Trojan attack. ProFlip finds the vulnerable bits in two iterations ($n_e = 2$) in all settings while n_b differs slightly. The results in the last three columns are the same since our CBS pipeline (Algorithm 2) identifies the same vulnerable elements and the same optimal values in these three cases.

Table 6: Effect of sample size on ProFlip's performance when attacking ResNet-18 with CIFAR-10 ($t = 2, TAP = 9.76\%$). The parenthesis in the last column shows the number of bit flips in each iteration of critical bits search.

Data Size	TA (%)	ASR (%)	$\mathbf{n_e}$	$\mathbf{n_b}$
64	89.0	95.7	2	8 (6+2)
128	88.4	96.3	2	7 (5+2)
256	90.3	97.9	2	12 (5+7)
512	90.3	97.9	2	12 (5+7)
1024	90.3	97.9	2	12 (5+7)

5. Discussion

Trade-off between Stealthiness and Effectiveness. ProFlip allows the adversary to explore the trade-off between Trojan stealthiness and effectiveness by setting the hyper-parameters γ_1 and γ_2 when computing $\mathcal{L}_{troj}$ in Eq. (10). We assess the trade-off between these two attack goals by changing γ_1 while using a fixed value of $\gamma_2 = 1$. Table 7 shows the evaluation results on ResNet-18 and CIFAR-10 dataset. It can be observed that the test accuracy of the Trojaned model increases as γ_1 grows, while the ASR shows a decreasing trend. We can also see that ProFlip's critical bits searching is *robust* to a wide range of γ_1, since the number of modified elements remains the same ($n_e = 2$) and the variation of n_b is small.

Table 7: Performance trade-off of ProFlip with varying hyper-parameters γ_1 in Trojan loss. ResNet-18 model with CIFAR-10 is assessed with $t = 2$, and $TAP = 9.76\%$.

γ_1	TA (%)	ASR (%)	n_e	n_b
1	89.44	97.68	2	11 (5+6)
2	89.44	97.68	2	11 (5+6)
4	90.30	97.88	2	12 (5+7)
8	90.38	96.31	2	12 (6+6)

Potential Defense. We propose a potential defense against ProFlip with two goals: (i) Reducing the ASR, and (ii) Increasing the BFA overhead in terms of n_b. We make a key observation from Figure 3 that selecting the most susceptible parameter for attack is crucial for BFA. As such, we propose to 'hide' the top vulnerable parameters of a DNN by performing *decomposition* on them. Existing matrix/tensor decomposition methods [44, 19, 45] can be used for this purpose. With the defense, the decomposed components are stored in memory instead of the raw parameter values. In this case, the adversary will attack the less vulnerable parameters that are stored in the raw format.

The overhead of our proposed defense depends on two factors: the complexity of the employed decomposition technique, and the number of layers selected for decomposition. As such, the defense overhead can be controlled by tuning these two factors. We implement this defense scheme by decomposing (thus protecting) the most vulnerable parameter identified by Algorithm 3. Table 8 compares ProFlip's performance before and after applying the defense. One can see that the proposed defense can effectively reduce the ASR of ProFlip while increasing the bit flip overhead n_b. We observe that the SVHN model with VGG-16 is more vulnerable compared to the other three, since its increase of n_b with defense is the smallest. However, our defense can increase n_b from 20 to 124 by decomposing the top-3 sensitive parameters of VGG-16.

Table 8: Performance of the proposed defense against ProFlip. The attack results before and after deploying the defense are denoted by 'bef.' and 'aft.', respectively. Termination condition for CBS is set to $n_e = 30$.

Dataset	Model	ASR (%)		$\mathbf{n_e}$		$\mathbf{n_b}$	
		bef.	aft.	bef.	aft.	bef.	aft.
CIFAR-10	ResNet-18	97.9	73.7	2	30	12	111
	VGG-16	95.4	90.4	4	30	18	128
SVHN	VGG-16	94.5	91.2	5	9	20	41
ImageNet	ResNet-18	94.3	67.1	3	30	15	127

6. Conclusion

We present ProFlip, the first practical, progressive bit flip-based targeted Trojan attack that can disturb a DNN after its deployment. ProFlip identifies the vulnerable bits in the model parameters with a gradual refinement of granularity, allowing the adversary to shrink the large search space efficiently. ProFlip outperforms the prior art in terms of both attack effectiveness and efficiency by yielding a higher attack success rate with fewer bit flips. Our attack engenders over 94% ASR across various benchmarks and reduces the number of bit flips by 31.8× on average compared to the previous work. ProFlip discloses the vulnerability of DNNs against bit-flip attacks at runtime and encourages the development of defense methods for model protection.

Acknowledgement

This work was supported by ARO (W911NF1910317), and SRC-Auto (2019-AU-2899).

References

[1] Cifar-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html. Accessed: 2021-03-13. 5

[2] Imagenet. http://image-net.org/. Accessed: 2021-03-13. 5

[3] The street view house numbers (svhn) dataset. http://ufldl.stanford.edu/housenumbers/. Accessed: 2021-03-13. 5

[4] Trojai. https://www.nist.gov/itl/ssd/trojai. Accessed: 2021-03-12. 3

[5] Zhezhi He Adnan Siraj Rakin and Deliang Fan. Tbt: Targeted neural network attack with bit trojan - cvpr2020. https://github.com/adnansirajrakin/TBT-CVPR2020, 2020. 6

[6] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to flip a bit? In *2010 IEEE 16th International On-Line Testing Symposium*, pages 235–239. IEEE, 2010. 2

[7] Scott Alfeld, Xiaojin Zhu, and Paul Barford. Data poisoning attacks against autoregressive models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. 1

[8] Jiawang Bai, Baoyuan Wu, Yong Zhang, Yiming Li, Zhifeng Li, and Shu-Tao Xia. Targeted attack against deep neural networks via flipping limited weight bits. *arXiv preprint arXiv:2102.10496*, 2021. 2

[9] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *IJCAI*, pages 4658–4664, 2019. 3

[10] Brice Colombier, Alexandre Menu, Jean-Max Dutertre, Pierre-Alain Moëllic, Jean-Baptiste Rigaud, and Jean-Luc Danger. Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 1–10. IEEE, 2019. 2

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 4

[12] Gaurav Dhiman, Raid Ayoub, and Tajana Rosing. Pdram: A hybrid pram and dram main memory system. In *2009 46th ACM/IEEE Design Automation Conference*, pages 664–669. IEEE, 2009. 2

[13] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 1

[14] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 245–261. IEEE, 2018. 2

[15] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019. 1, 2, 3

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4

[17] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraș. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 497–514, 2019. 2, 4, 5

[18] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014. 2, 5

[19] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015. 8

[20] Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al. Adversarial examples in the physical world, 2016. 1

[21] Shuangchen Li, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. Drisa: A dram-based reconfigurable in-situ accelerator. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 288–301. IEEE, 2017. 2

[22] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International conference on machine learning*, pages 2849–2858. PMLR, 2016. 2

[23] Geert Litjens, Clara I Sánchez, Nadya Timofeeva, Meyke Hermsen, Iris Nagtegaal, Iringo Kovacs, Christina Hulsbergen-Van De Kaa, Peter Bult, Bram Van Ginneken, and Jeroen Van Der Laak. Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific reports*, 6(1):1–11, 2016. 1

[24] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. 2017. 1, 2

[25] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 45–48. IEEE, 2017. 1

[26] Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Weller. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. International Joint Conferences on Artificial Intelligence, Inc., 2017. 1

[27] Szymon Migacz. 8-bit inference with tensorrt. In *GPU technology conference*, volume 2, page 5, 2017. 2

[28] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38, 2017. 1

[29] Onur Mutlu and Jeremie S Kim. Rowhammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(8):1555–1571, 2019. 2, 5

[30] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016. 3

[31] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1211–1220, 2019. 2, 4, 5

[32] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Tbt: Targeted neural network attack with bit trojan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13198–13207, 2020. 2, 3, 4, 5, 6, 7

[33] Adnan Siraj Rakin, Zhezhi He, Jingtao Li, Fan Yao, Chaitali Chakrabarti, and Deliang Fan. T-bfa: Targeted bit-flip adversarial weight attack. *arXiv preprint arXiv:2007.12336*, 2020. 2

[34] Mohammad Samragh, Mojan Javaheripi, and Farinaz Koushanfar. Codex: Bit-flexible encoding for streaming-based fpga acceleration of dnns. *arXiv preprint arXiv:1901.05582*, 2019. 2

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 4

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 4

[37] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1675–1689, 2016. 2, 5

[38] Matthew Veres and Medhat Moussa. Deep learning for intelligent transportation systems: A survey of emerging trends. *IEEE Transactions on Intelligent transportation systems*, 21(8):3152–3168, 2019. 1

[39] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019. 3

[40] Rey Wiyatno and Anqi Xu. Maximal jacobian-based saliency map attack. *arXiv preprint arXiv:1808.07945*, 2018. 3

[41] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016. 2

[42] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 1463–1480, 2020. 2

[43] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019. 1

[44] Qingchen Zhang, Laurence T Yang, Xingang Liu, Zhikui Chen, and Peng Li. A tucker deep computation model for mobile multimedia feature learning. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 13(3s):1–18, 2017. 8

[45] Zhisheng Zhong, Fangyin Wei, Zhouchen Lin, and Chao Zhang. Ada-tucker: Compressing deep neural networks via adaptive dimension adjustment tucker decomposition. *Neural Networks*, 110:104–115, 2019. 8

[46] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 2